CPEN 311: Digital Systems Design

Finite State Machines (Control Circuits)

Learning Objectives

- Distinction between datapath and control
  - Control is usually built using finite state machine (FSM)
- Difference between Moore machine and Mealy machine
- How to design Moore machine
- How to design Mealy machine

FSM Design Summary

Manually creating an FSM involves the following 6 steps:

1. Generate state diagram
2. Generate state table
3. State assignment
4. Truth tables
5. Karnaugh maps and logic equations
6. Circuit diagram

Finite State Machines

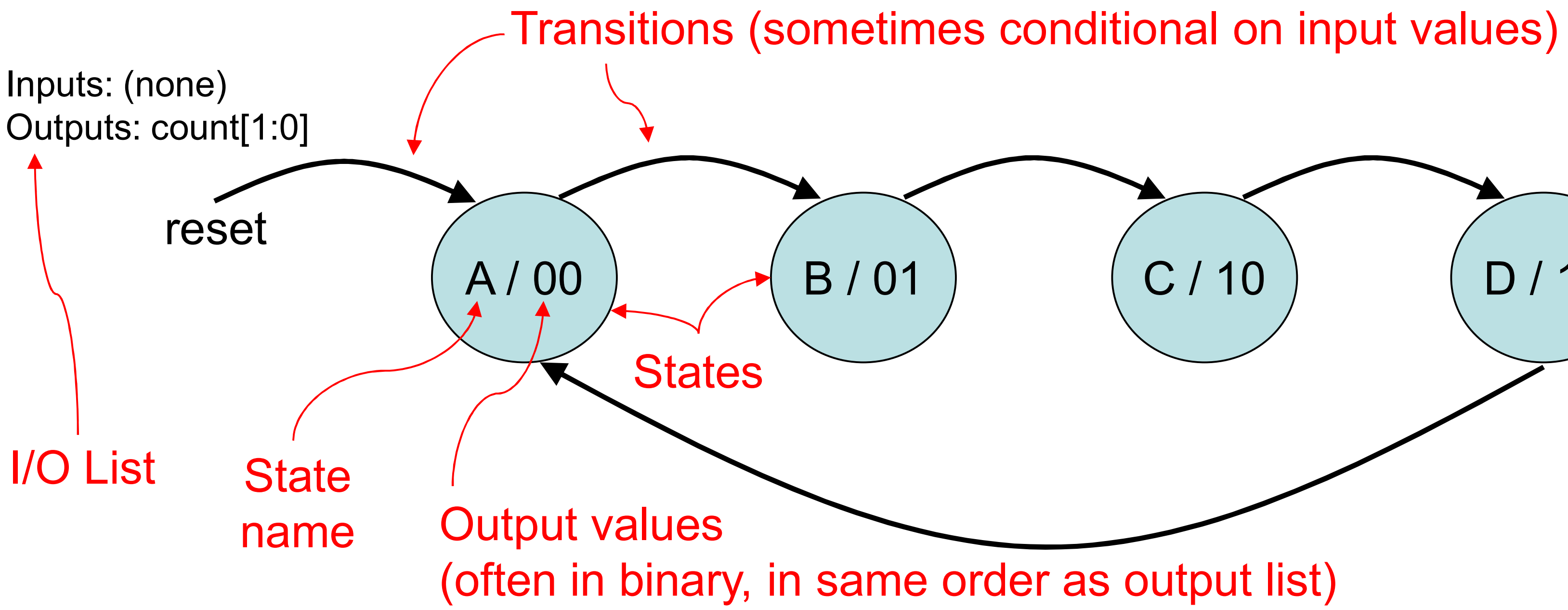
- Logic circuits generally divided into two parts
  - Datapath that does the “arithmetic”
    - adders, ALUs, muxes, registers
  - Control logic
    - Arranges the “sequencing” or steering of data in datapath
    - Often designed as a finite state machine (FSM)

- Finite State Machine
  - A “program” or set of “instructions” for the datapath
  - Eg: “do step 1, do step 2, move data here, if(…) repeat”

- Example FSM: 2-bit Counter (0, 1, 2, 3, 0, 1, …)
  - After reset: count = 0
  - Each clock cycle: count = count + 1

Example FSM 1

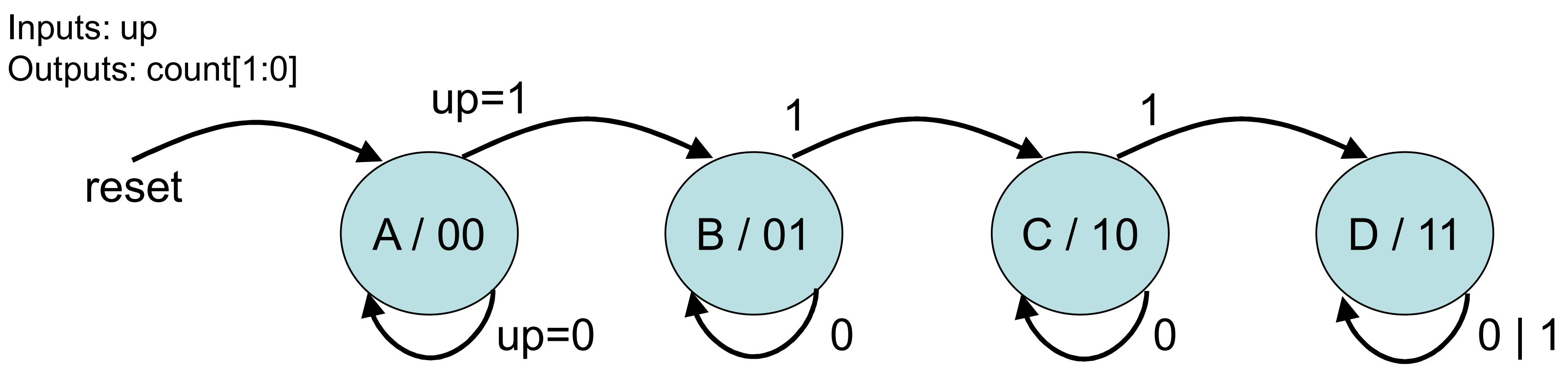
- 2-bit Counter (0, 1, 2, 3, 0, 1, 2, …)
  - Reset: count = 0. Each cycle: count++.



- Only 1 state active at a time
  - Eg: state A is active for 1 cycle after reset, state B is active for 1 cycle after A, etc.
  - Always designate a "reset" state
- Transitions occur on every clock edge
  - Each state always has an outward transition (can be to self)
  - Transition can be conditional upon an input value

Example FSM 2

- 2-bit Conditional Saturating Counter (0, 1, 2, 3, 3, …)
  - Reset: count = 0. Each cycle: if(up==1 & count<3) count++.



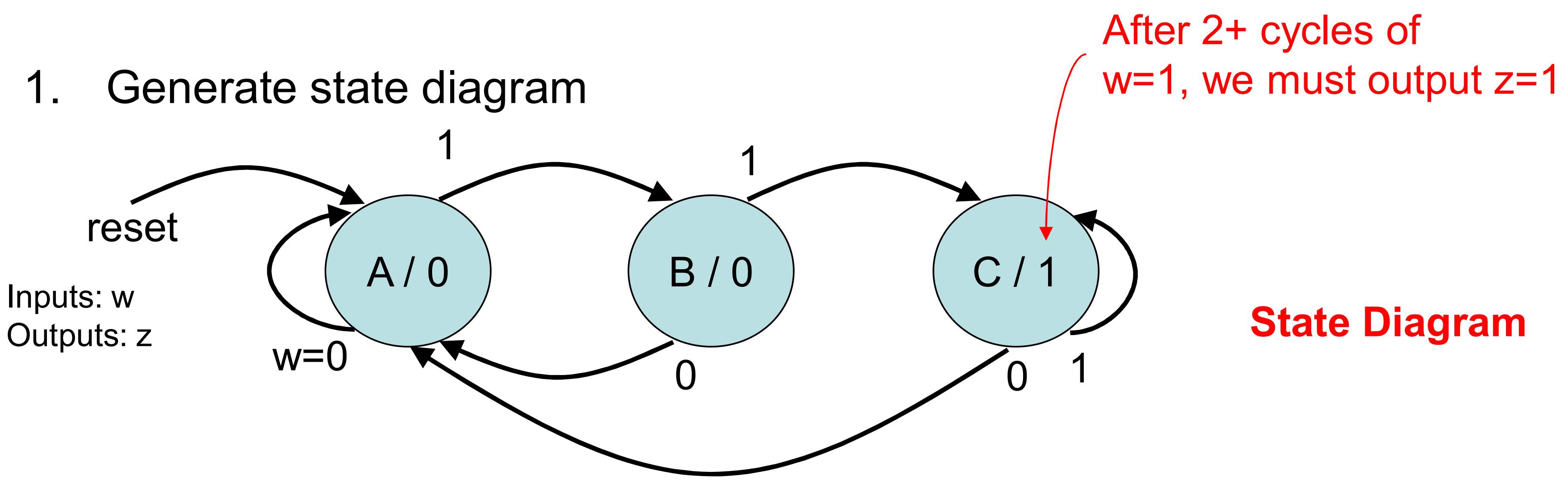
- Transitions labeled with conditions
  - up=0: transition to same state
  - up=1: transition to next state
- For each state
  - Outgoing transitions for **all possible input values**
    - If any input value is omitted, what should be the next state?
  - Outgoing transitions have **mutually exclusive conditions**
    - Can only follow 1 outgoing transition (“deterministic finite automata” or DFA)
- Like outputs, inputs can have multiple signals or multiple bits
  - Clearly define an order

Example: FSM as Logic

Example Problem

- Develop a circuit with input **w**, output **z**, reset, clock
- Output **z**=1 if previous 2 clock cycles both see w=1, otherwise z=0

1. Generate state diagram



State Diagram

2. Generate state table

Present State	Next State W=0	Next State W=1	Outputs Z
A*	A	B	0
B	A	C	0
C	A	C	1

* Reset state

State Table

Deriving Logic Equations

Given State Table, derive logic equations

Present	**Next State**		**Outputs**
State	**w=0**	**w=1**	**z**
A*	A	B	0
B	A	C	0
C	A	C	1

State Table (before state assignment)

3. State assignment: assign binary values to states
   - A=00, B=01, C=10 (other assignments possible, eg A=10, B=01, C=11)

Present	**Next State**		**Outputs**
State	**w=0**	**w=1**	**z**
00*	00	01	0
01	00	10	0
10	00	10	1
11	xx	xx	x

State Table (after state assignment)

- Notice 11 isn't used by any state, but we must add it to table
  - Why?
  - Use xx = don't care values. Why?

Deriving Logic Equations

4. Derive truth tables after state assignment
   - PS = present state, signals y2 y1
   - NS = next state, signals Y2 Y1

PS	**NS: Y2 Y1**		**Outputs**
y2 y1	**w=0**	**w=1**	**z**
00*	00	01	0
01	00	10	0
10	00	10	1
11	xx	xx	x

State Table (after state assignment)

Truth Tables

Inputs	**Output Y2**	
y2 y1	**w=0**	**w=1**
00	0	0
01	0	1
10	0	1
11	x	x

Inputs	**Output Y1**	
y2 y1	**w=0**	**w=1**
00	0	1
01	0	0
10	0	0
11	x	x

Inputs	**Output**
y2 y1	**z**
00	0
01	0
10	1
11	x

(ignore reset for now; don’t forget that reset must initialize y2y1 = 00)

Deriving Logic Equations

5. Derive logic equations and Karnaugh maps from truth tables
   - PS = present state, signals y2 y1
   - NS = next state, signals Y2 Y1

Inputs	Output Y2	
y2 y1	w=0	w=1
00	0	0
01	0	1
10	0	1
11	x	x

Inputs	Output Y1	
y2 y1	w=0	w=1
00	0	1
01	0	0
10	0	0
11	x	x

Inputs	Output
y2 y1	z
00	0
01	0
10	1
11	x

Truth Tables

Signal Y2

y2y1 \ w	0	1
00	0	0
01	0	1
11	x	x
10	0	1

Signal Y1

y2y1 \ w	0	1
00	0	$\boxed{1}$
01	0	0
11	x	x
10	0	0

Signal z

y2 \ y1	0	1
0	0	0
1	1	x

Karnaugh Maps

Y2 = w & y1 + w & y2
= w & (y1 + y2)

Y1 = w & !y2 & !y1

z = y2

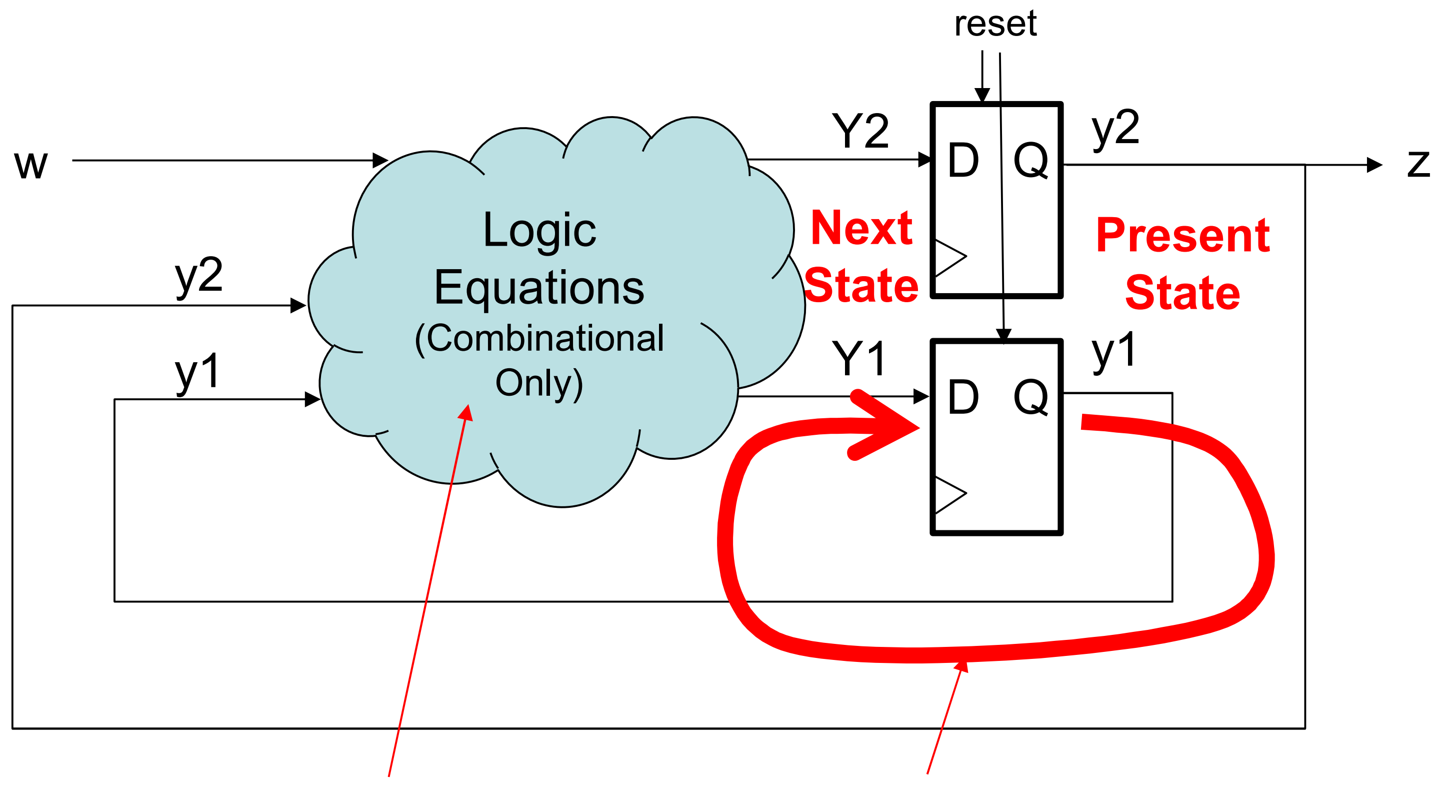
Logic Equations

Notice 1: next state Y2, Y1 is a function of present state y2, y1 and input w
Notice 2: output z is a function of present state y2 (not input w)

Circuit Diagram

6. Derive circuit diagram from logic equations

Y2 = w & (y1 + y2)   Y1 = w & !y2 & !y1   z = y2



Logic Equations → Circuit

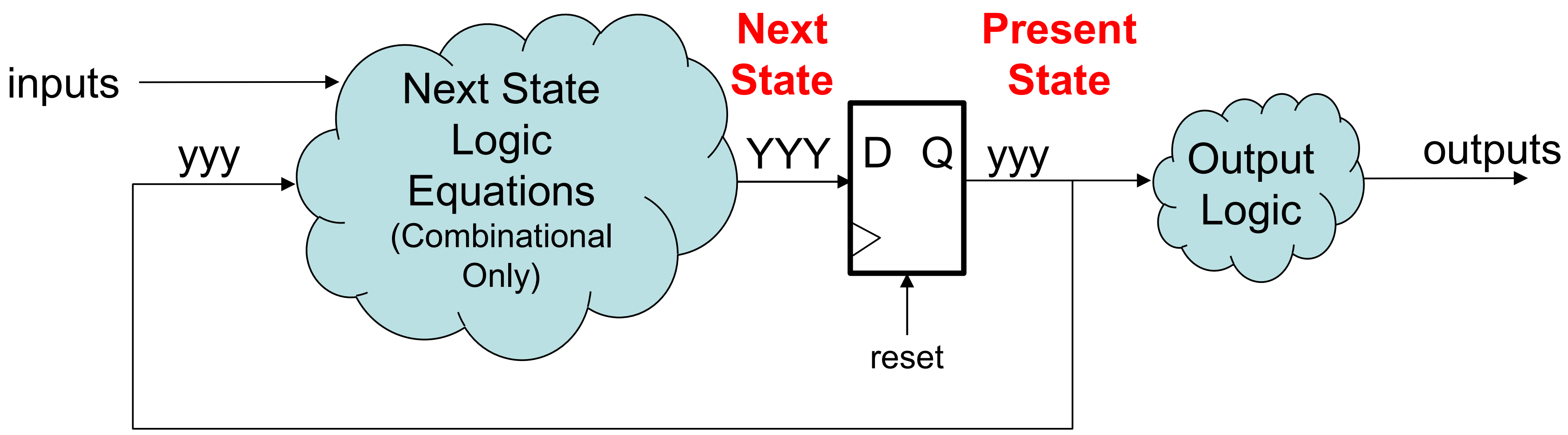
Combinational logic never has loops

Sequential logic can have loops!

Why and how does this work?

1) Clock adds time delay from y1 → Y1
2) Both y1 and y2 change at same time (on clock edge); Y2, Y1 stable before edge

General FSM Model: Moore Machine



General model of **Moore Machine** FSM model

- Outputs are **only** a function of present state
    - **By design** because we wrote output values inside state bubbles
    - Inputs never appear in equation for outputs

FSM Design Summary

Manually creating an FSM involves the following 6 steps:

1. Generate state diagram
2. Generate state table
3. State assignment
4. Truth tables
5. Karnaugh maps and logic equations
6. Circuit diagram

You DO NOT want to enter the circuit diagram in Verilog

- Instead, you can replace steps 2..6 with:

2. Generate state-transition code in Verilog

Quartus will happily synthesize (& optimize) the required logic for you

Be sure to follow the <u>Moore model</u> (outputs depend on state only)!

Moore FSM Example

Example Problem

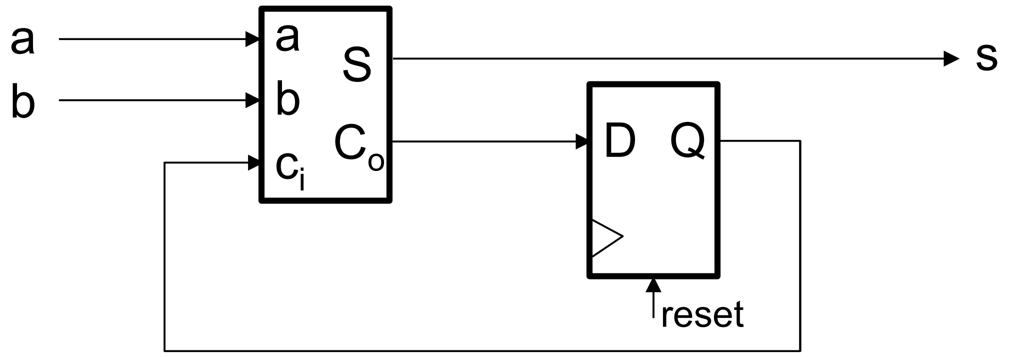
- Design the Moore FSM for a serial adder (implies reset, clock)
- Inputs **a**, **b** presented serially (LSB first, 1 bit per clock cycle)
- Output **s** is **a**+**b**, presented serially (LSB first)

Full adder:

$$\begin{array}{r} a \\ b \\ +\ c_i \\ \hline C_o\ \ S \end{array}$$

S = a XOR b XOR c_i

$C_o = a \,.\, b + a \,.\, c_i + b \,.\, c_i$



We intuitively know the solution…

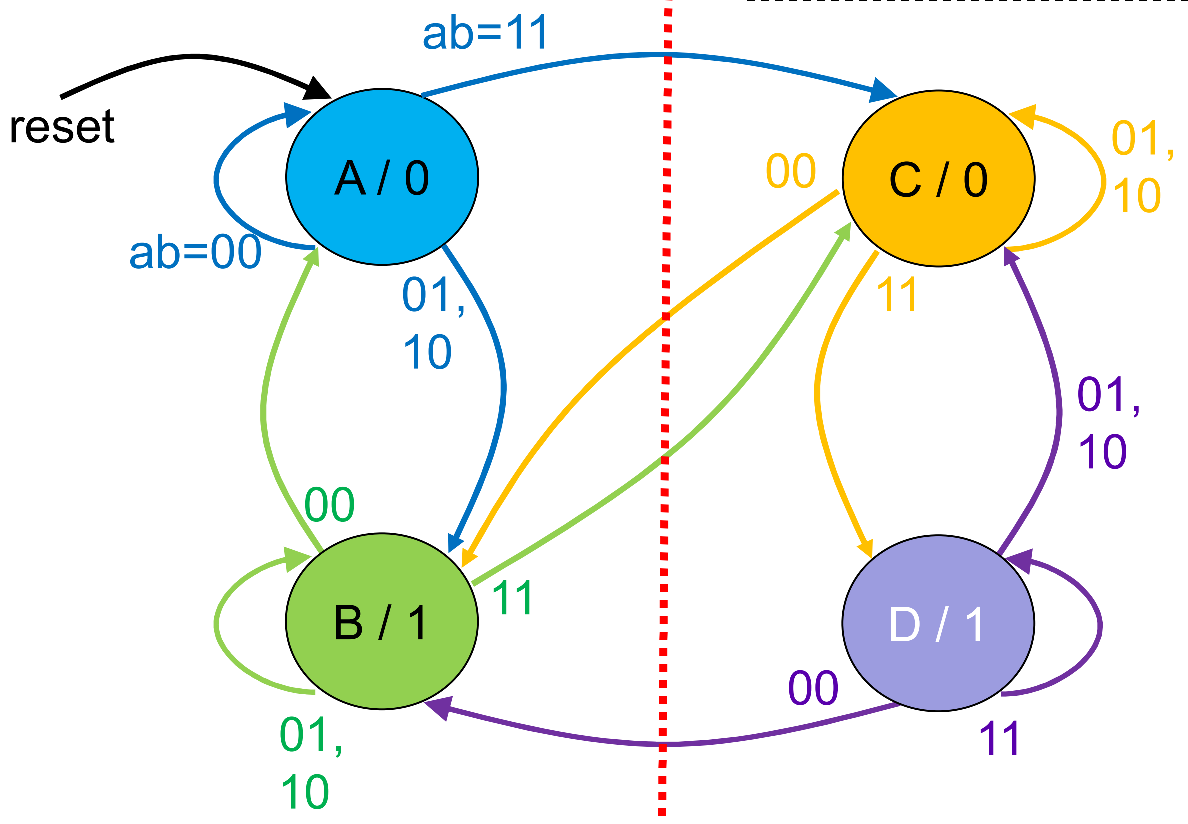
- Full adder to compute sum (S) and carry-out (C_o)
- Use a flip-flip to connect carry-out to carry-in on the next clock cycle
- Output **s** is function of inputs **a**, **b** and present state ➔ not a Moore FSM

How to derive **__as Moore FSM__**?

Moore FSM for Serial Adder

Inputs: a, b
Outputs: s

Comment: this solution has 4 states, so it requires 2 flip-flops. Our intuitive solution on previous slide needed only 1 flip-flop. **Why?**



State A, C
Output s=0

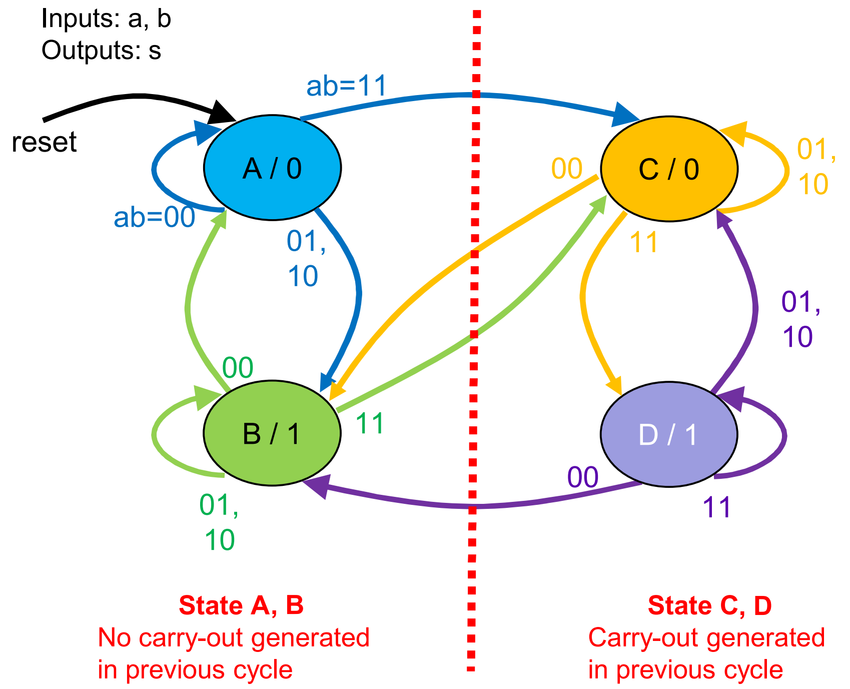
State B, D
Output s=1

State A, B
No carry-out generated in previous cycle

State C, D
Carry-out generated in previous cycle

CHECK:
For each state, do outgoing edges cover all 4 combinations of ab inputs (00, 01, 10, 11)?

Moore FSM for Serial Adder



Present State	Next State ab=00	01	11	10	Output s
A*	A	B	C	B	0
B	A	B	C	B	1
C	B	C	D	C	0
D	B	C	D	C	1

* Reset state

State assignment:

A=00, B=01, C=10, D=11

Present State y2y1	Next State Y2Y1 ab=00	01	11	10	Output s
00	00	01	10	01	0
01	00	01	10	01	1
11	01	10	11	10	1
10	01	10	11	10	0

STEPS:

1. State diagram
2. State table
3. State assignment
4. Truth table
5. K-map and logic
6. Circuit

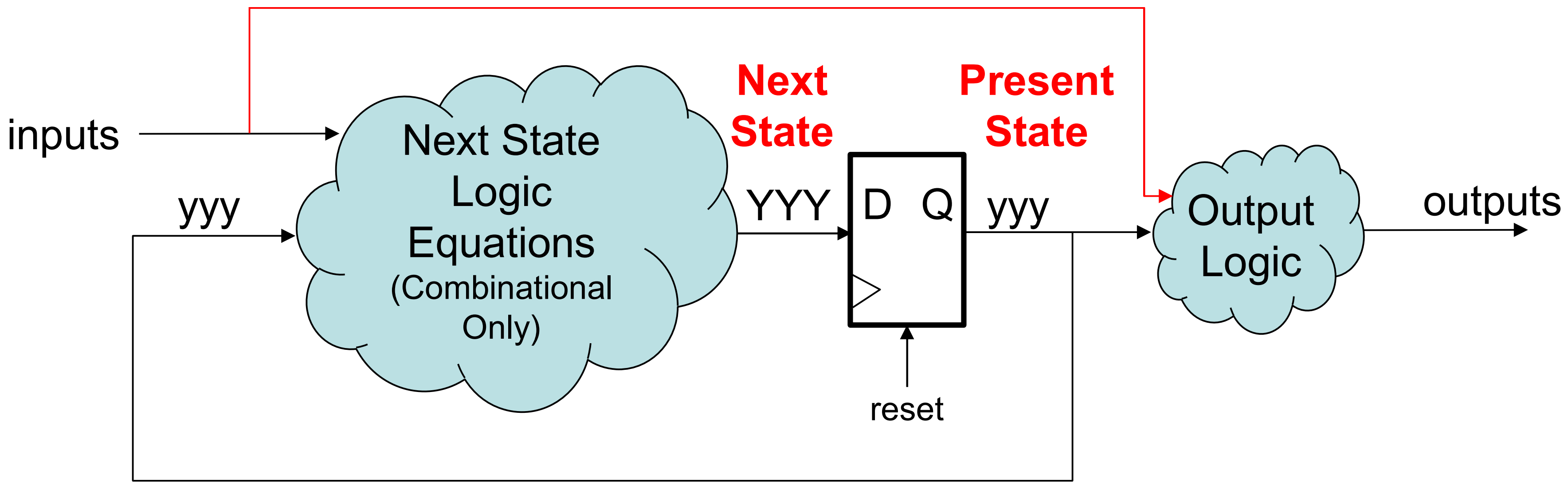
Y1 = a XOR b XOR y2

Y2 = a.b + a.y2 + b.y2

s = y1

- Verify the above!
- Notice that c_i = y2
- Then, if we remove FF for y1, we get the intuitive solution!

General FSM Model: Mealy Machine



General model of **<u>Mealy Machine</u>** FSM model

- Outputs are a function of **<u>both</u>** state **<u>and</u>** inputs
  - **<u>By design</u>** because we will write output values on transitions
  - Changes to inputs may cause immediate change to outputs
- Typical properties of Mealy…
  - has fewer states than Moore (why?)
  - logic for outputs is more complex, produces glitchy outputs
  - next-state logic is more complex, slower clock frequency
  - often takes fewer clock cycles to do a task

Mealy FSM Example

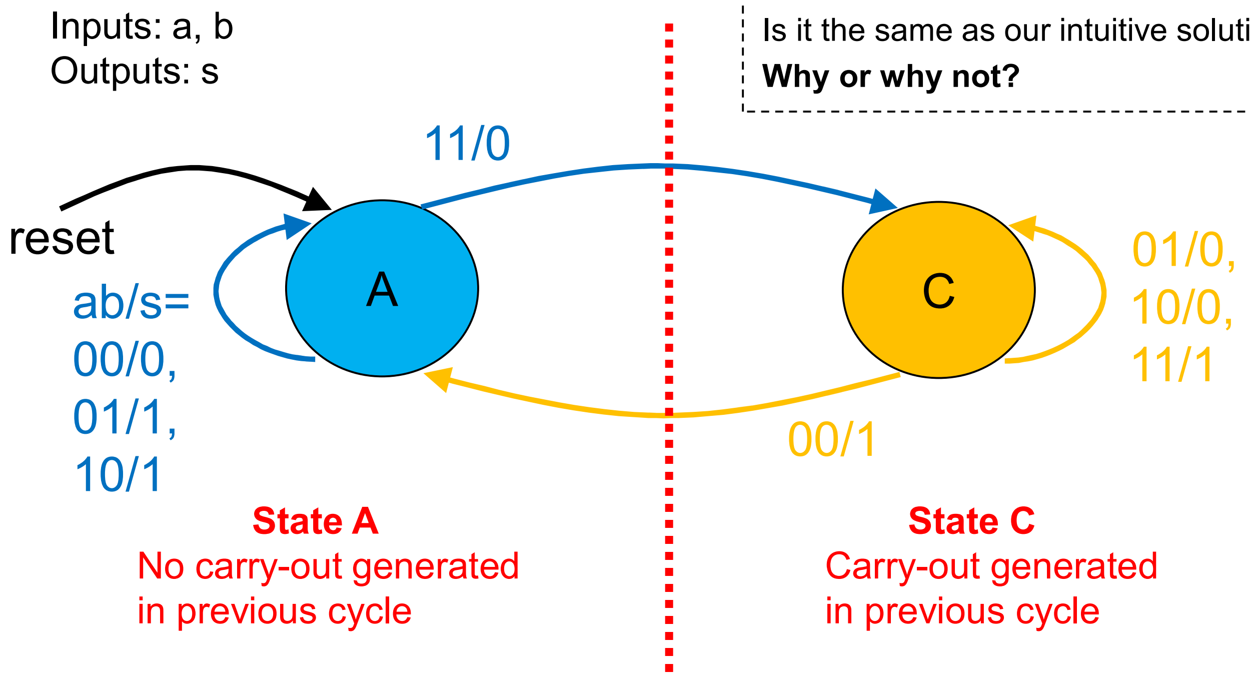
Example Problem

- Design the Mealy FSM for a serial adder (implies reset, clock)
- Inputs **a**, **b** presented serially (LSB first, 1 bit per clock cycle)
- Output **s** is **a+b**, presented serially (LSB first)

(this is the same as before, except use Mealy model instead of Moore)

Mealy FSM for Serial Adder

Comment: this solution has 2 states, so it requires 1 flip-flop.
Is it the same as our intuitive solution?
Why or why not?



STEPS:

1. State diagram
2. State table
3. State assignment
4. Truth table
5. K-map and logic
6. Circuit

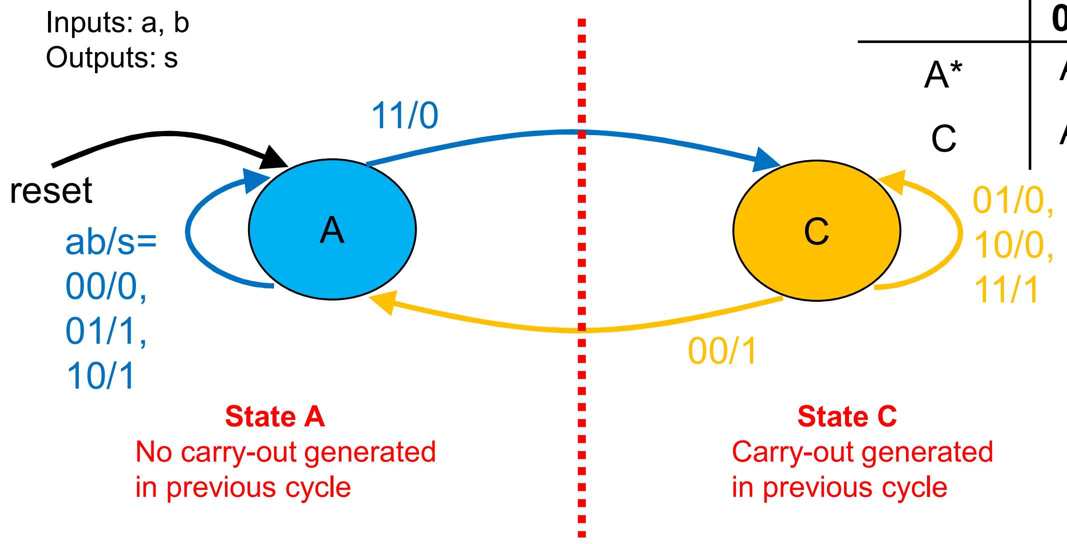
Notice

For each state, all possible input values are labelled on transitions.

For each possible input value, in each state, an output value is specified.

Output values can change when staying in the same state.

Mealy FSM Example



Present State	Next State ab=00	01	11	10	Output s ab=00	01	11	10
A*	A	A	C	A	0	1	0	1
C	A	C	C	C	1	0	1	0

State assignment: A=0, C=1

STEPS:

1. State diagram
2. State table
3. State assignment
4. Truth table
5. K-map and logic
6. Circuit

$Y2 = a.b + a.y2 + b.y2$

$s = a \text{ XOR } b \text{ XOR } y2$

- Verify the above!
- Notice that $c_i = y2$
- This is the same as the intuitive solution!

Present State y2	Next State Y2 ab=00	01	11	10	Output s ab=00	01	11	10
0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	0	1	0

Mealy vs. Moore

- Moore is usually easier to design
  - But it usually has more states in total
  - But it can take more states to complete a task
    - **Moore** has **more** states ← way to remember difference
  - Outputs are delayed until next clock cycle
  - Outputs are stable (no glitches) entire clock cycle
- Mealy advantages
  - Outputs can change immediately (glitches)
  - Fewer states in total, since outputs can depend upon inputs
  - Completes a task in fewer clock cycles
    - Faster than Moore?
  - But next-state logic can be more complex (slower clock speed)
    - Slower than Moore?
- Hybrids are possible…
  - Some outputs function of state only
  - Some outputs function of state+inputs

- Serial adder examples: Mealy solution looks simpler than Moore
  - Only because it's a trivial example
  - Usually opposite is true!

Summary

- Use FSMs to control a datapath
  - Always sequential logic
    - Has memory of past inputs!
    - Has loops (new state is function of current state)
  - Moore machine $\Leftarrow$ preferred method for novices
    - Outputs function of state only
    - More total states (one state for every possible output value)
    - Outputs stable entire clock cycle
  - Mealy machine
    - Outputs function of state and inputs
    - Fewer total states (output values on transition conditions instead of states)
    - Fewer states to finish a task (outputs produced sooner)
    - Outputs may glitch, change when inputs change
- Design process
  - Two steps with Verilog
  - Six steps when done manually

Exercises 1

1. Design a Moore FSM that has an input **w** and an output **z**. The machine is a sequence detector that produces **z**=1 when the previous two values of **w** were 00 or 11; otherwise **z**=0.

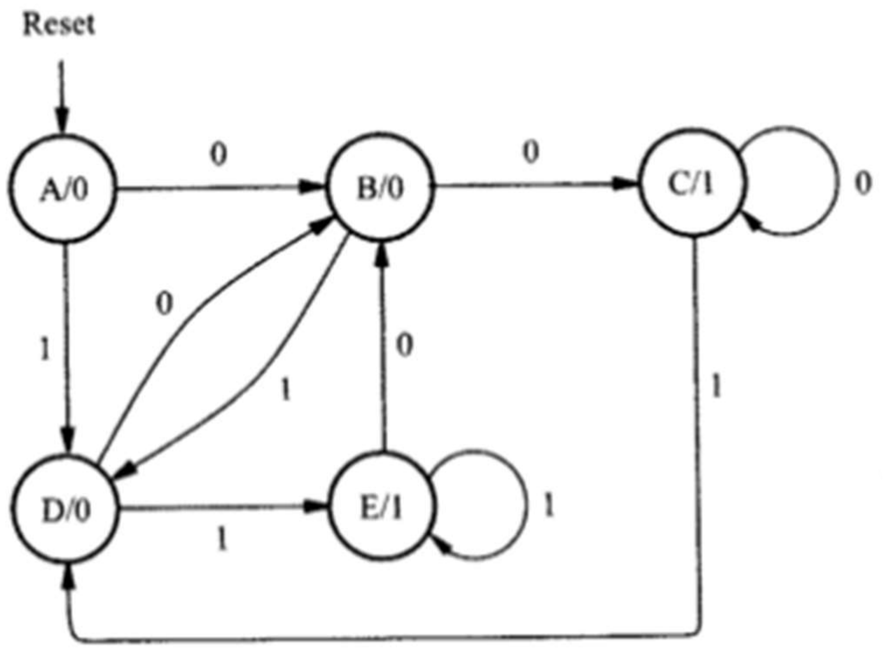
2. Write the Verilog code for the Moore FSM.

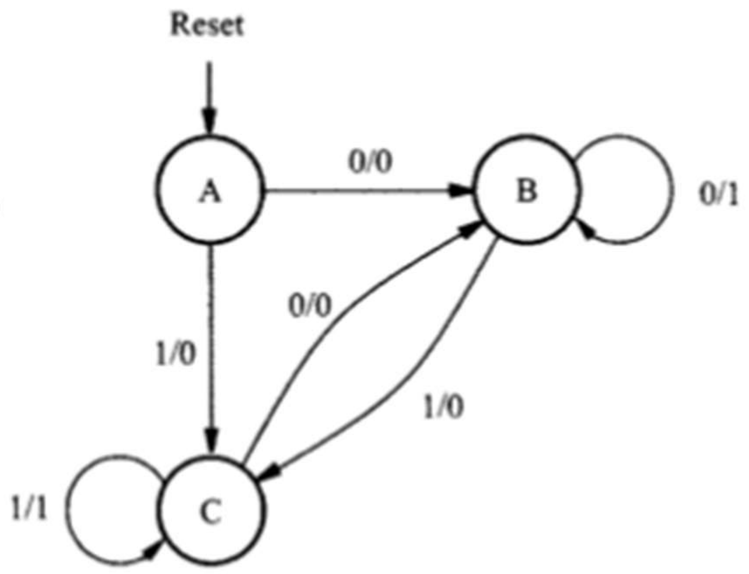
3. Repeat the design for a Mealy FSM.

4. Write the Verilog code for the Mealy FSM.

Exercises 2

5. Complete the design process for the following two FSMs. Which one is a Mealy machine? Which one is a Moore machine?


Reset
A/0
B/0
C/1
D/0
E/1
0
0
0
1
0
1
0
1
1
1


Reset
A
B
C
0/0
0/1
0/0
1/0
1/0
1/1